Profiling Deep Learning Workloads

Murali Emani,
Data Science group, ALCF
memani@anl.gov

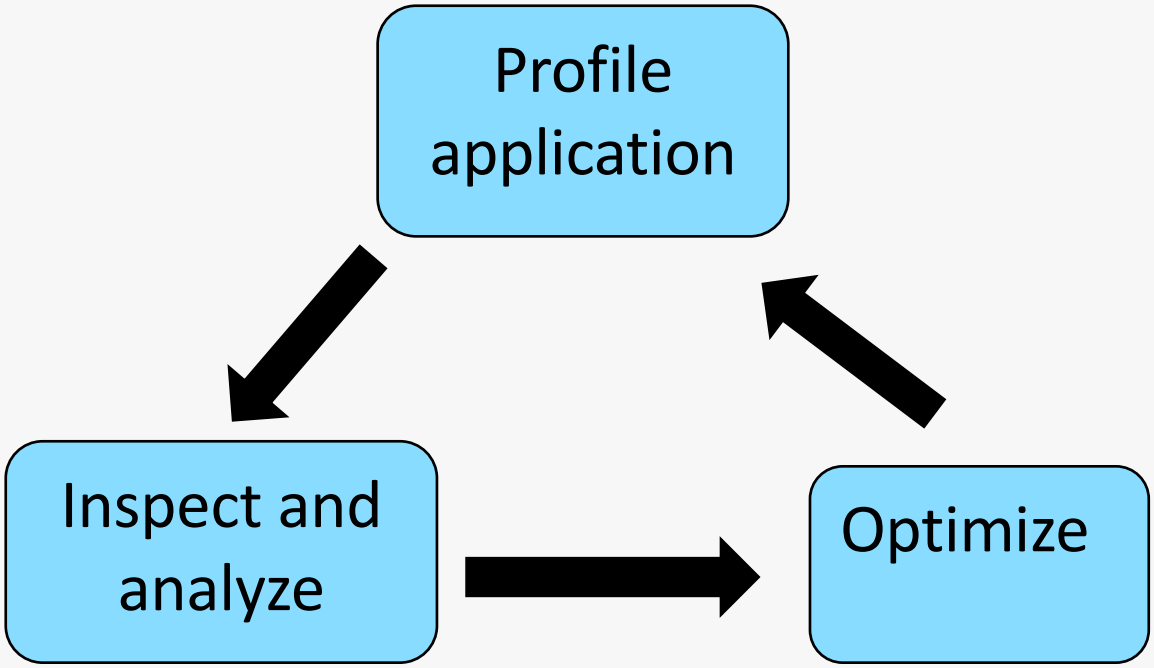
Introduction

- Profiling is an approach to measure application performance

- Simple Profiling:
  - How long does an application take
- Advanced Profiling:
  - Why does an operation take long time

- Goal: Find performance bottlenecks
  - inefficient programming
  - memory I/O bottlenecks
  - parallel scaling

Typical Optimization Workflow



Iterative workflow till desired performance is reached

Broad classification

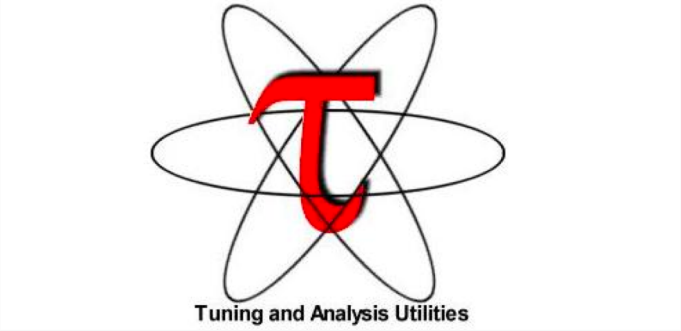
- Hardware counters
  - count events from CPU perspective (# of flops, memory loads, etc.)
  - usually needs Linux kernel module installed or root permission
- Statistical profilers (sampling)
  - interrupt program at given intervals to find the state of a program
- Event based profilers (tracing)
  - collect information on each function call

Plethora of Tools

- Cprofile
- Gprof
- Perf tool
- Intel Vtune
- HPCToolKit
- OpenSpeedShop
- TAU
- Nvidia Nvprof, Nsight

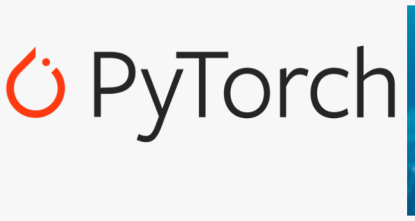
….

…



Profiling DNN workloads

- Critical to understand workload performance
- Machine learning and deep learning models are implemented on a variety of hardware
- Most applications are written in Python using standard ML frameworks

- The frameworks generate kernels based on hardware and customized installation and libraries (MKL-DNN, CuDNN etc.)

Challenges

- Profiling is hard, cumbersome and time-consuming
- Profiling tools generate lot of data and hard to understand
- The problem is further compounded with large, complex models with large volumes of data
- Need strategies to use right tools and detailed insights to how to analyze the profile data

Profiling on Nvidia GPUs

Profiling on Nvidia GPUs

Use Nvidia profiler '**Nvprof**'

- capture metrics from hardware counters
- invoked via command line or UI (Nvidia Visual Profiler NVVP)

See list of options using

`nvprof -h`

Some useful options:

`-o`: create output file to import into nvvp

`--metrics / -m` : collect metrics

`--events / -e` : collect events

`--log-file` : create human readable output file

`--analysis-metrics` : collect all metrics to import into nvvp

`--query-metrics/--query-events`: list of available metrics/events

Events and Metrics

- An **event** is a countable activity, action, or occurrence on a device. It corresponds to a single hardware counter value which is collected during kernel execution
- A **metric** is a characteristic of an application that is calculated from one or more event values

In general, events are only for experts, rarely used.

- Vary in number based on hardware family (P100, K80, V100 etc)
- For example, on V100, nvprof gives 175 metrics
- Event and metric values are aggregated across all units in the GPU.

Workflow – on Cooley

Option 1)

- Use '**nvprof'** to collect metrics in an output file (compute node)
- Use '**nvvp**' to visualize the profile (login node)

Option 2)

- Directly launch **nvvp** on compute node and profile the code interactively

Profile Commands

– Kernel timing analysis:

```
nvprof --log-file timing.log <myapp>
nvprof --log-file timing.log python myapp.py args
```

– Traces (#threads, #warps, #registers)

```
nvprof --print-gpu-traces --log-file traces.log <myapp>
```

– Get all metrics for all kernels

```
nvprof --metrics all --log-file all-metrics.log <myapp>
```

– Get metrics for guided analysis

```
nvprof --analysis-metrics -o analysis.nvvp <myapp>
```

– Visual profile to use Nvidia Visual Profiler (nvvp)

```
nvprof -o analysis.nvvp <myapp>
```

Selective Profiling

- As profiling adds significant overhead, a better strategy is to profile only regions of interest (kernels and metrics)

- All metrics for kernels of interest:

```
nvprof --profile-from-start off --kernels <kernel-name> --metrics all
--log-file selective-profile.log <myapp>
```

- few metrics for kernels of interest

```
nvprof --profile-from-start off--kernels <kernel-name> --metrics ipc
--log-file selective-profile.log <myapp>
```

For example, if we want to profile heavy kernels only

Step 1) use nvprof to list all kernels sorted by the time

Step 2) re-run nvprof in selective profiling mode

- Profile GEMM kernels

```
nvprof --profile-from-start off --kernels "::gemm:1" --metrics all
--log-file selective-profile.log <myapp>
```

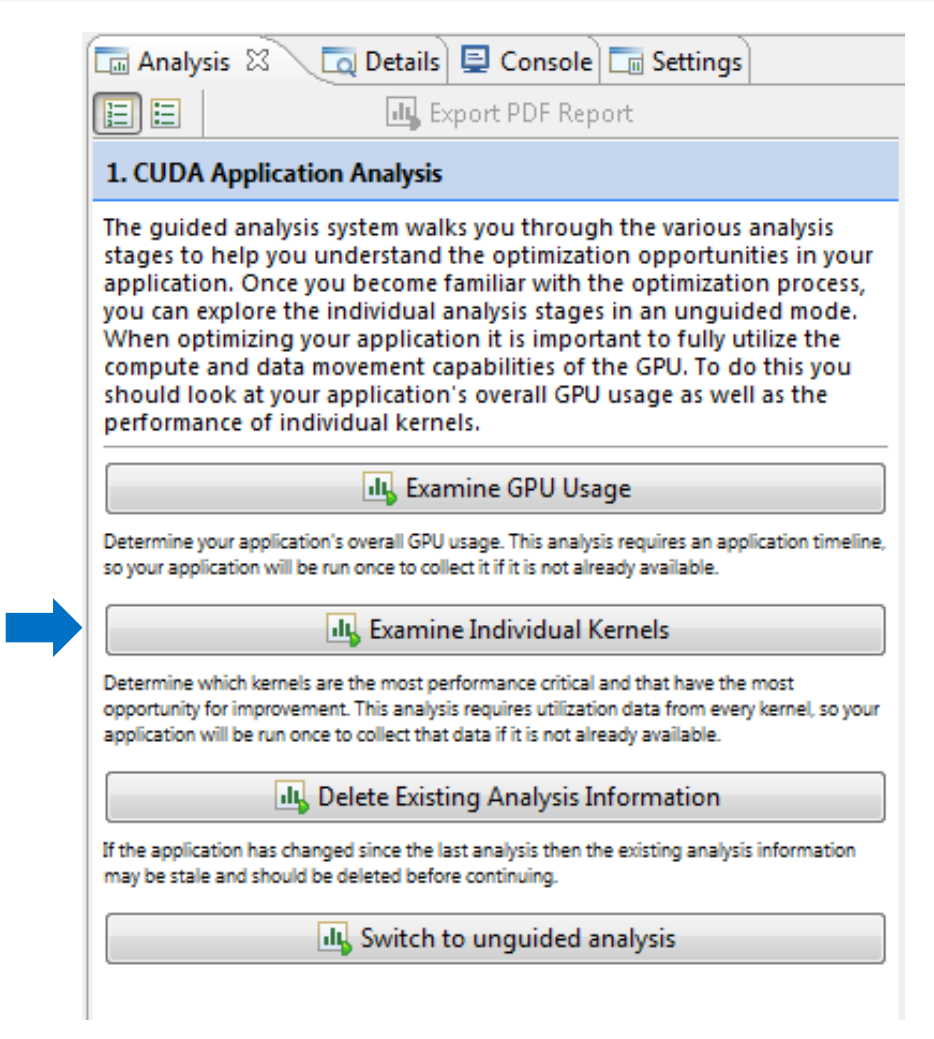
Metrics and Events

Metrics relevant to identify compute, memory, IO characteristics

achieved_occupancy	ratio of the average active warps per active cycle to the maximum number of warps supported on a multiprocessor
ipc	Instructions executed per cycle
gld_efficiency	Ratio of requested global memory load throughput to required global memory load throughput expressed as percentage.
gst_efficiency	Ratio of requested global memory store throughput to required global memory store throughput expressed as percentage.
dram_utilization	The utilization level of the device memory relative to the peak utilization on a scale of 0 to 10

Detailed Analysis

Use visual profiler nvvp



Kernel Optimization Priorities

The following kernels are ordered by optimization importance based on execution time and achieved occupancy. Optimization of higher ranked kernels (those that appear first in the list) is more likely to improve performance compared to lower ranked kernels.

Rank	Description
100	[2 kernel instances] maxwell_sgemm_128x64_tn
1	[1 kernel instances] elementWise(float*, float*, float*, float*, float*, float*)

Example

Simple CNN in Keras

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(…....)

model.fit(.....)
```

https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py

```
======== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   36.04%  46.6474s    202752  230.07us  124.57us  339.61us  void sgemm_largek_lds64<bool=0, boo
nt, int, int, int, int, float const *, float const *, float, float, int, int, int*, int*)
                   4.90%  6.34396s      5642  1.1244ms  52.928us  1.2132ms  void fermiPlusCgemmLDS128_batched<b
nst *, float2* const *, float2* const *, float2*, float2 const *, float2 const *, int, int, int, int, int, int,
                   4.60%  5.95521s      6562  907.53us  252.41us  1.4053ms  cgemm_strided_batched_sm35_ldg_nt_6
                   4.22%  5.46673s     13778  396.77us  34.912us  536.64us  void tensorflow::BiasNCHWKernel<flo
                   3.44%  4.44991s      6889  645.94us  185.95us  748.12us  void sgemm_largek_lds64<bool=0, boo
nt, int, int, int, int, float const *, float const *, float, float, int, int, int*, int*)
                   3.35%  4.33847s     12200  355.61us  43.871us  473.31us  void fft2d_c2r_32x32<float, bool=0,
t, int, int, int, float, float, cudnn::reduced_divisor, bool, float*, float*, int2, int, int)
                   3.19%  4.12895s     12208  338.22us  27.552us  464.35us  void fft2d_r2c_32x32<float, bool=0,
, cudnn::reduced_divisor, bool, int2, int, int)
```

```
  API calls:   47.93%  42.3065s      6891  6.1394ms  4.2120us  8.8403ms  cuCtxSynchronize
               26.73%  23.5916s   2092524  11.274us  4.8920us  40.285ms  cudaLaunchKernel
                5.37%  4.74056s         8  592.57ms  1.9750us  4.74054s  cudaStreamCreateWithFlags
                5.32%  4.69823s    209098  22.469us     545ns  4.29630s  cudaPointerGetAttributes
                3.81%  3.36602s    356228  9.4490us  4.1710us  34.767ms  cudaMemcpyAsync
                2.37%  2.09162s    231818  9.0220us     358ns  9.9905ms  cuEventRecord
                1.37%  1.20962s     66998  18.054us  5.8700us  6.1906ms  cuMemcpyHtoDAsync
                1.29%  1.14020s    833508  1.3670us     438ns  2.1696ms  cuEventQuery
```

Nvidia Nsight Tools

- **Nsight Systems** - System-wide application algorithm tuning
- **Nsight Compute** – Debug CUDA API and optimize CUDA kernels

- To profile

```
$ nsys profile python train.py
```

- This generates profile file in ‘report.qdrep’ which can be imported to view with Nsight Systems UI

- To identify which kernels are run on Tensorcores (dedicated HW units for half/mixed precision matrix multiply-accumulate ops)

```
$ nv-nsight-cu-cli --kernel-id ::s884:1 python train.py
```

NSIGHT SYSTEMS

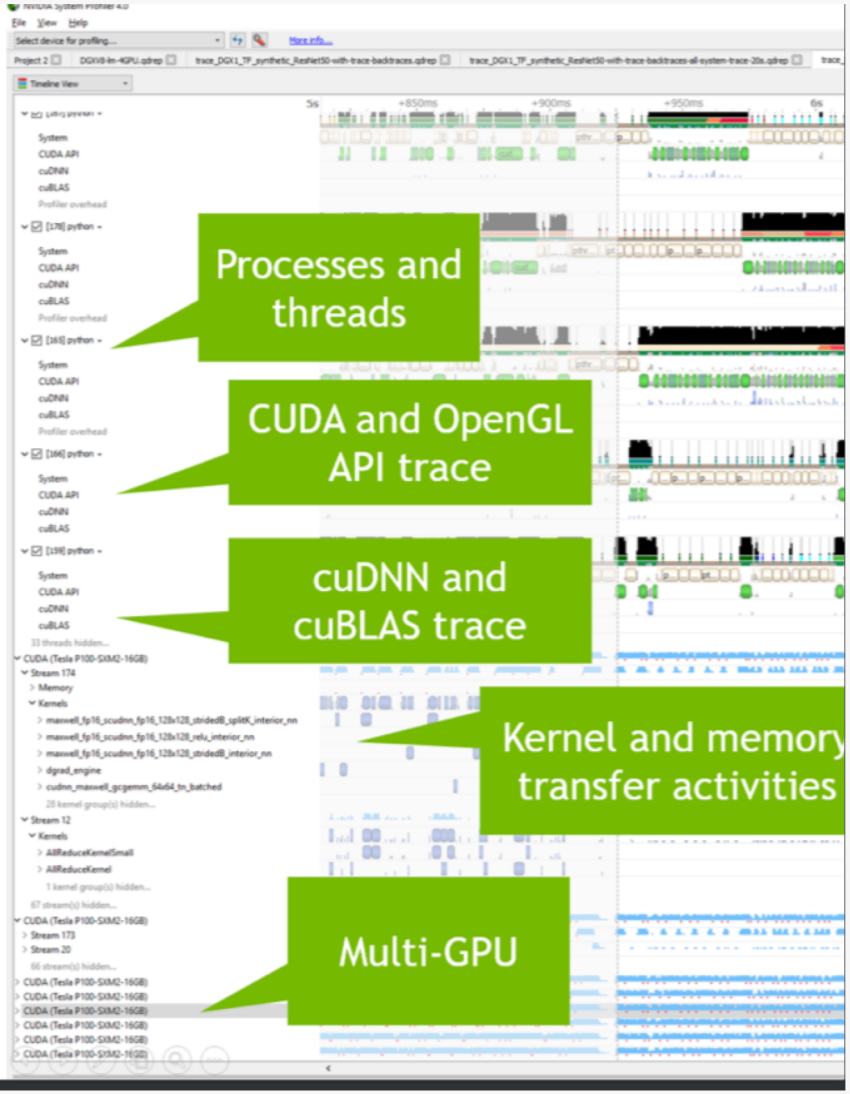
Next-Gen System Profiling Tool

System-wide application algorithm tuning
- Multi-process tree support

Locate optimization opportunities
- Visualize millions of events on a fast GUI timeline
- Or gaps of unused CPU and GPU time

Balance your workload across multiple CPUs and GPUs
- CPU algorithms, utilization, and thread state
- GPU streams, kernels, memory transfers, etc

Multi-platform: Linux & Windows, x86-64 & Tegra, MacOSX (host only)



https://bluewaters.ncsa.illinois.edu/liferay-content/document-library/content/NVIDIA%20Nsight%20Systems%20Overview%20by%20Sneha%20Kottapalli.pdf

NSIGHT COMPUTE

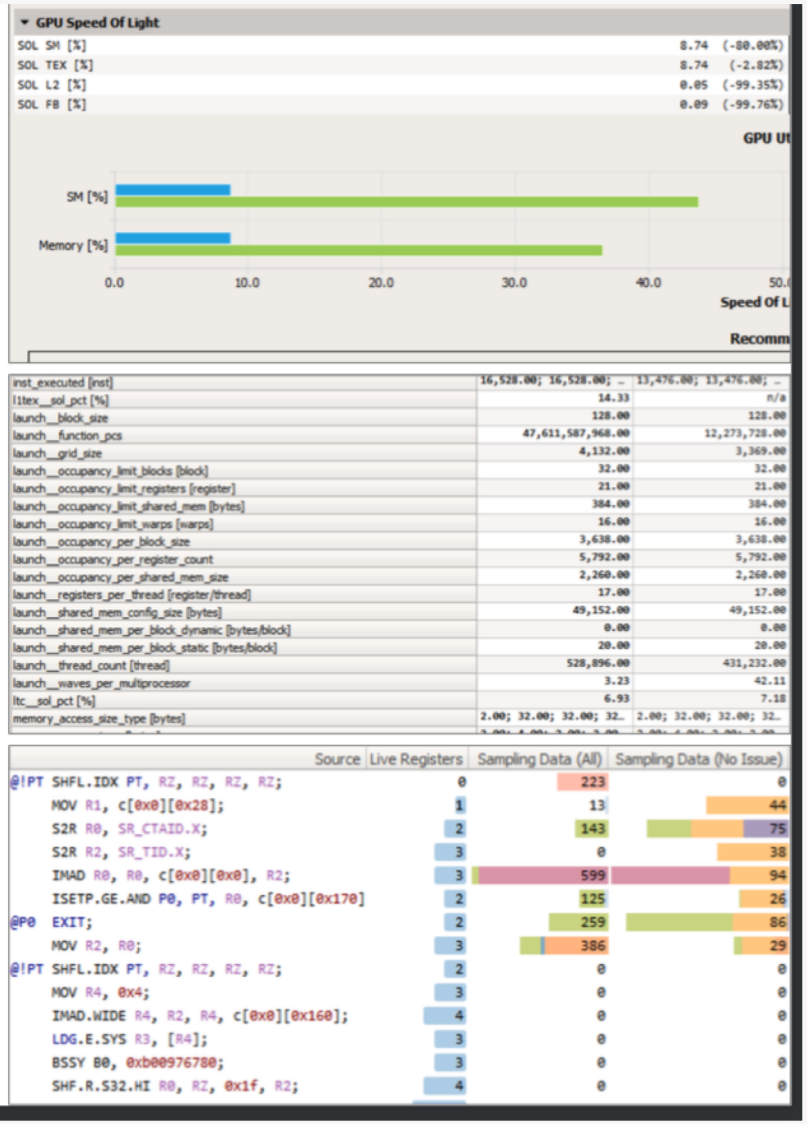
Next-Gen Kernel Profiling Tool

Key Features:

- Interactive CUDA API debugging and kernel profiling
- Fast Data Collection
- Improved Workflow (diffing results)
- Fully Customizable (programmable UI/Rules)
- Command Line, Standalone, IDE Integration

OS: Linux, Windows, ARM, MacOSX (host only)

GPUs: Pascal (GP10x), Volta, Turing



https://bluewaters.ncsa.illinois.edu/liferay-content/document-library/content/NVIDIA%20Nsight%20Systems%20Overview%20by%20Sneha%20Kottapalli.pdf

Argonne NATIONAL LABORATORY

Profiling on CPUs using Intel Vtune

Application Performance Snapshot (APS)

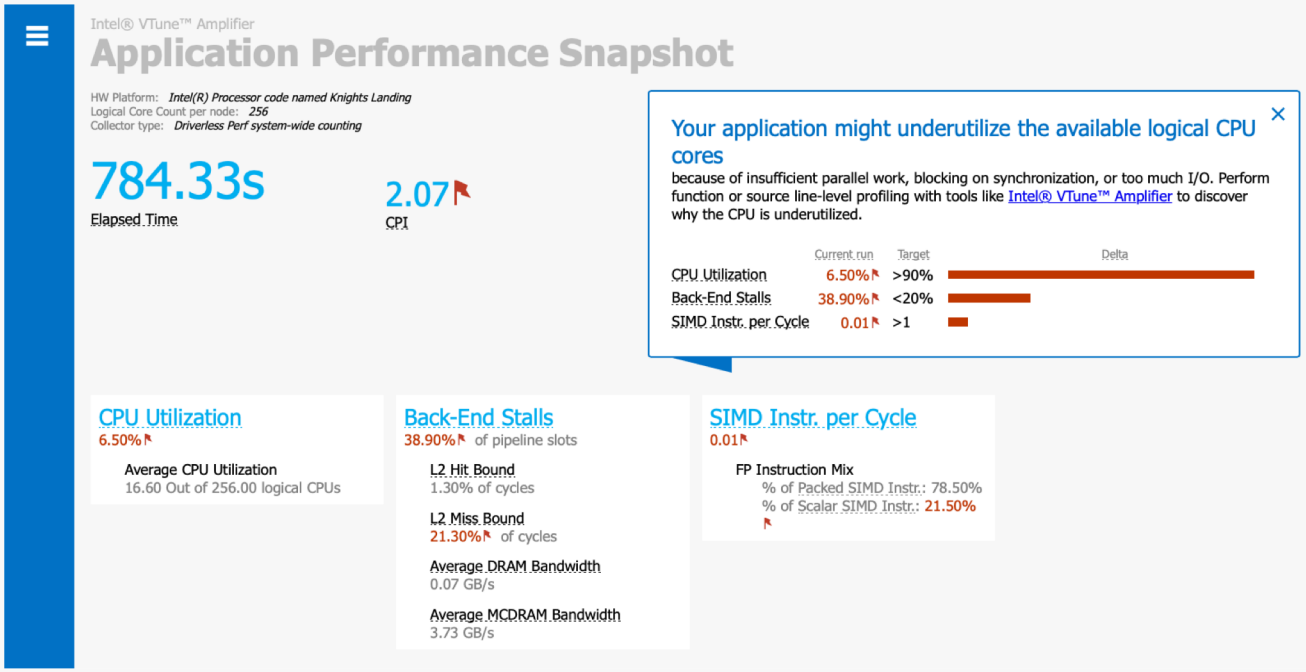
APS generates a highlevel performance snapshot of your application. Easy to run:

```
source /soft/compilers/intel/19.0.3.199/vtune_amplifier/apsvars.sh
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/soft/compilers/intel/19.0.3.199/vtune_amplifier
/lib64
aps --result-dir=aps_results/ -- python /full/path/to/script.py
```

Results can be viewed in a single html file, or via command line:

```
| Summary information
|-----------------------------------------------------------------
  HW Platform                : Intel(R) Processor code named Knights Landing
  Logical core count per node: 256
  Collector type             : Driverless Perf system-wide counting
  Used statistics            : aps_results
|
| Your application might underutilize the available logical CPU cores
| because of insufficient parallel work, blocking on synchronization, or too much I/O.
Perform function or source line-level profiling with tools like Intel(R) VTune(TM)
Amplifier to discover why the CPU is underutilized.
  CPU Utilization:                                  6.50%
| Your application might underutilize the available logical CPU cores because of
| insufficient parallel work, blocking on synchronization, or too much I/O.
| Perform function or source line-level profiling with tools like Intel(R)
```

Application Performance Snapshot (APS)



Pros

- Very easy to use
- Tracks important hardware metrics:
  - Thread Load Balancing
  - Vectorization
  - CPU Usage

Cons

- Only high level information – but then again, that is the design of this tool.

Intel Vtune – Hotspots

`sampling-mode=sw` - User-Mode Sampling (default) used for profiling:

- Targets running longer than a few seconds
- A single process or a process-tree
- Python and Intel runtimes

`sampling-mode=hw` - (Advanced hotspots) Hardware Event-Based Sampling used for profiling:

- Targets running less than a few seconds
- All processes on a system, including the kernel

Intel Vtune – Advanced Hotspots

Advanced Hotspots analysis

- Detailed report of how effective the computation is on CPUs
- extends the hotspots analysis by collecting call stacks, context switch and statistical call count data and analyzing the CPI (Cycles Per Instruction) metric.

```
amplxe-cl -collect hotspots -knob sampling-mode=hw -finalization-mode=none -r vtune-result-dir_advancedhotspots/ -- python /full/path/to/script.py
```

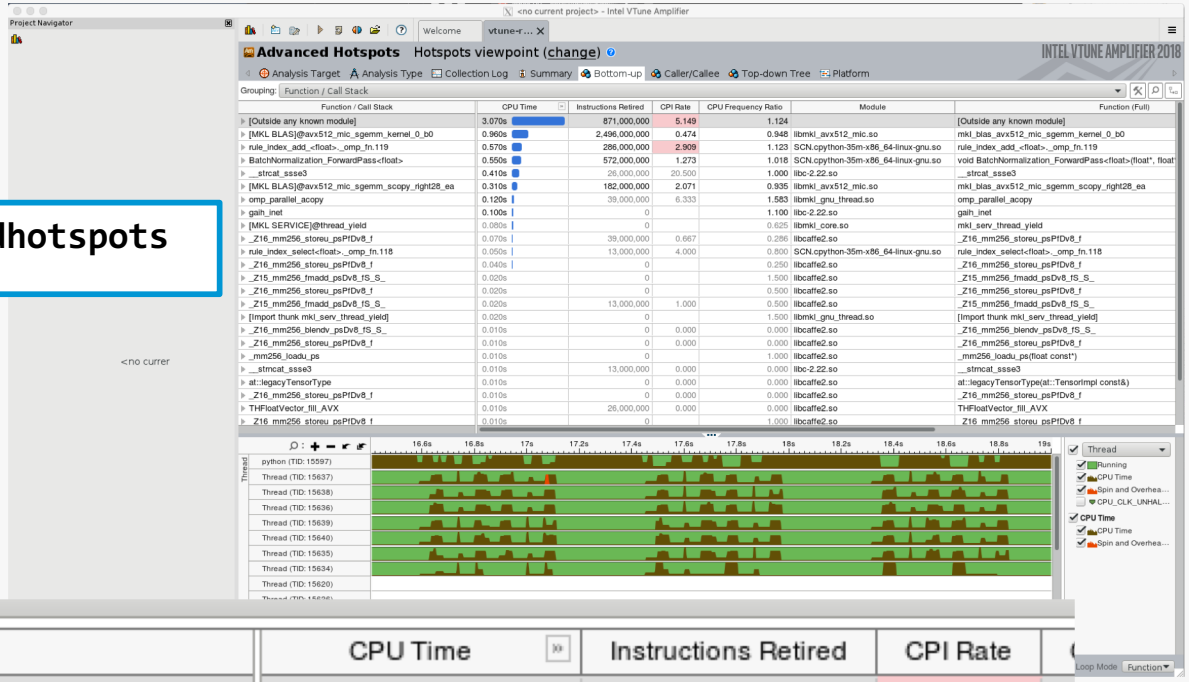
Run the finalization step after the run completes from the login nodes

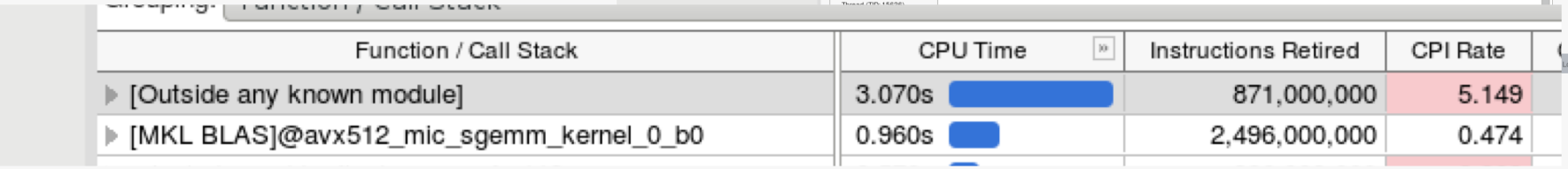
```
amplxe-cl -finalize -search-dir / -r vtune-result-dir_advancedhotspots
```

Intel Vtune – Advanced Hotspots

Run the GUI to view your results:

```
amplxe-gui  vtune-result-dir_advancedhotspots
```





- Visualize each thread activity and the functions that cause it.
- Give a bottom up and top down view, very useful for seeing which functions are hotspots

Useful Commands

```
amplxe-cl -c hotspots -- python3 myapp.py
amplxe-cl -R hotspots -report-output report-hotspots.csv -format csv

amplxe-cl -c uarch-exploration -k sampling-interval=100 -- python3 myapp.py
amplxe-cl -R uarch-exploration -report-output report-uarch-exploration.csv -format csv

amplxe-cl -c memory-access -k sampling-interval=100 -- python3 myapp.py
amplxe-cl -R memory-access -report-output report-memory-access.csv -format csv

amplxe-cl -c memory-consumption -k sampling-interval=100 -- python3 myapp.py
amplxe-cl -R memory-consumption -report-output report-memory-consumption.csv -format csv

change sampling interval
-k sampling-interval=<number>
```

Useful Commands

```
amplxe-cl -report hw-events/summary -r r000ue/ -report-output ./report-uarch.csv -format
csv

amplxe-cl -collect hotspots -strategy ldconfig:notrace:notrace -- python myapp.py

## get MKL-DNN verbose
export MKLDNN_VERBOSE=2
amplxe-cl -collect hotspots -strategy ldconfig:notrace:notrace -- python myapp.py
```

Hands-on Exercise

Example scripts to profile an image classification CNN model with TF/Keras

```
https://github.com/argonne-lcf/ATPESC_MachineLearning
cd Profiling
```

Cooley

```
qsub -A training -q training -t 1:00:00 -n 1 qsub_mnist_profile_gpu.sh
```

Theta

```
qsub -A ATPSEC2020 -q ATPSEC2020 -t 1:00:00 -n 1 qsub_mnist_profile_cpu.sh
```


Thank you!

backup

Operations on backward weights, data have stalls → high memory requirements

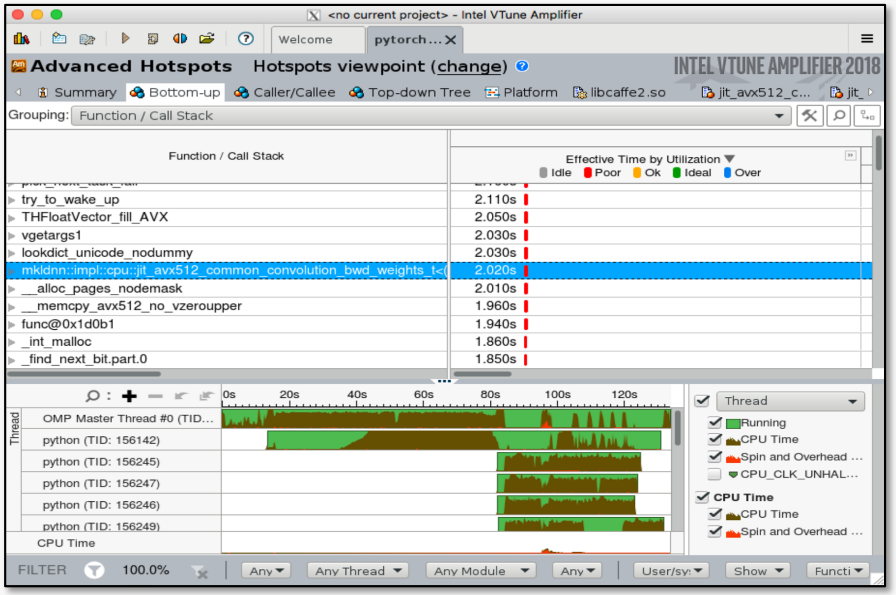
– Convolution layer is sensitive to compute units, memory and cachelines
– Dense layer is sensitive to communication -> bandwidth

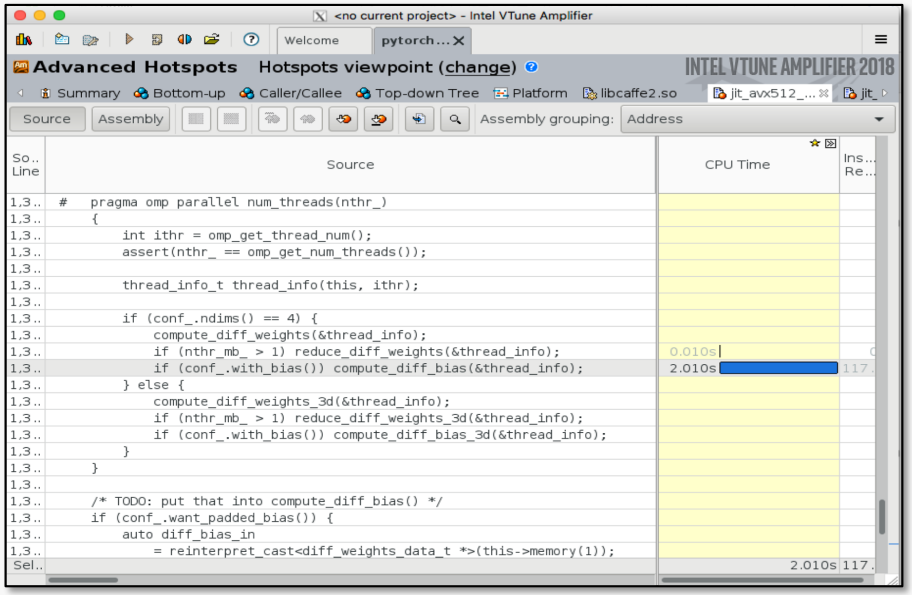
VTune profiling

More details: Profiling Your Application with Intel VTune and Advisor - Carlos Rosales-Fernandez and Paulius Velesko, Intel

```
source /opt/intel/vtune_amplifier/amplxe-vars.sh
aprun -n … -e OMP_NUM_THREADS=128 \
        -e LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/vtune_amplifier/lib64 \
        ampxle-cl -collect advance-hotspots -r output_dir python script.py
```

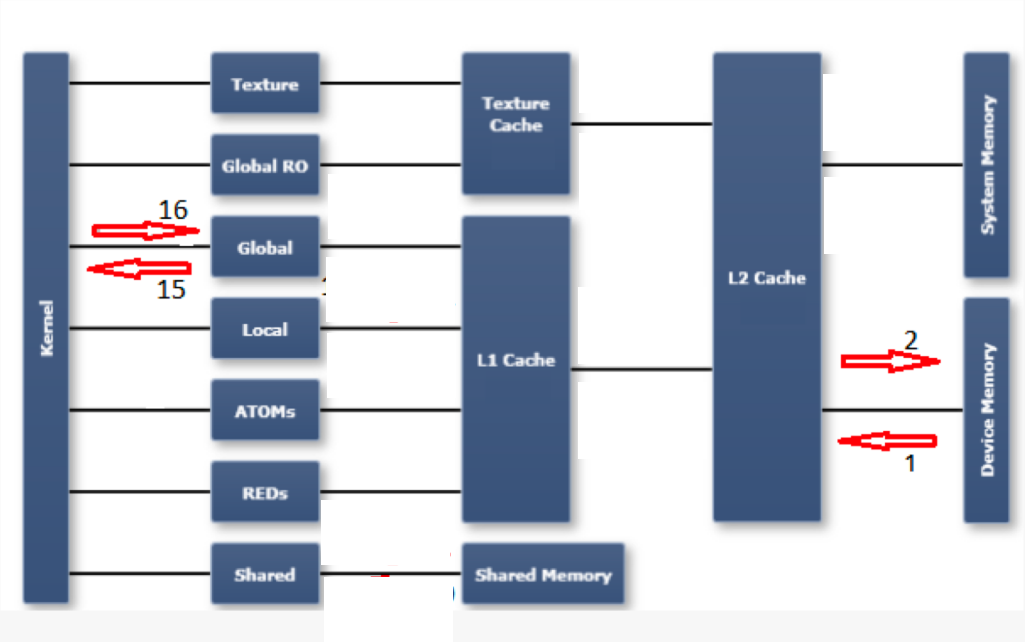
Remember to set LD_LIBRARY_PATH,
Put vtune library at the end!! Otherwise, it might complaint about the GLIBCXX version.





The python modules are compiled using -g flag. Therefore, the user could trace the source file in Vtune.

GPU Memory - metrics



GPU Memory

1.dram_read_throughput, dram_read_transactions
2.dram_write_throughput, dram_write_transactions
3.sysmem_read_throughput, sysmem_read_transactions
4.sysmem_write_throughput, sysmem_write_transaction
5.l2_l1_read_transactions, l2_l1_read_throughput
6.l2_l1_write_transactions, l2_l1_write_throughput
7.l2_tex_read_transactions, l2_texture_read_throughput
8.texture is read-only, there are no transactions possible
this path
9.shared_load_throughput, shared_load_transactions
10.shared_store_throughput, shared_store_transactions
11.l1_cache_local_hit_rate
12.l1 is write-through cache, so there are no (independen
metrics for this path - refer to other local metrics
13.l1_cache_global_hit_rate
14.see note on 12
15.gld_efficiency, gld_throughput, gld_transactions
16.gst_efficiency, gst_throughput, gst_transactions

https://stackoverflow.com/questions/37732735/nvprof-option-for-bandwidth

GEMM – 2*m*n*k operations

m, k – hidden layer size

n = minibatch size

2 * 512 * 512 * 64 = 0.03 GFLOP

Peak upper limit = 6000 GFLOP/s

Runtime ~ 5.6 usec

Time(%)	Time	Calls	Avg	Min	Max	Name
93.93%	575.72us	8	**71.964us**	70.241us	78.945us	maxwell_sgemm_128x64_tn

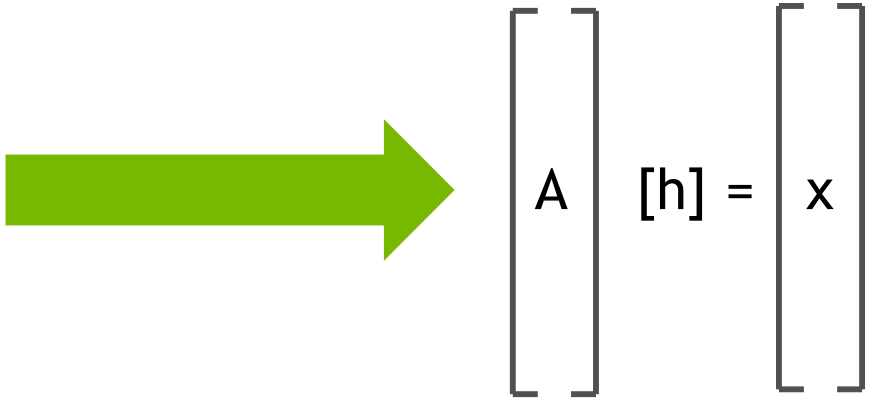
Optimization

$[A_1][h] = [x_1]$

$[A_2][h] = [x_2]$

$[A_3][h] = [x_3]$

$[A_4][h] = [x_4]$




```
Time(%)      Time      Calls       Avg       Min       Max  Name
 93.93%  575.72us          8  71.964us  70.241us  78.945us  maxwell_sgemm_128x64_tn
```

```
Time(%)      Time      Calls       Avg       Min       Max  Name
 84.40%  198.11us          2  99.057us  98.177us  99.937us  maxwell_sgemm_128x64_tn
```

2.5x performance gain